Complete Cloudflare Tunnel Setup Tutorial for Ubuntu 22.04 Headless Homelab Server

This tutorial guides you through setting up a Cloudflare Tunnel using the Zero Trust dashboard (web portal) to securely expose Docker app containers running on a headless Ubuntu 22.04 (Jammy Jellyfish) server. The setup focuses on production use with HTTPS-only services, ensuring encrypted traffic from the edge to your origin. We'll proxy traffic to Docker containers (e.g., via localhost:<port>) without exposing your server's public IP.

The instructions are based on official Cloudflare documentation for creating dashboard-managed tunnels, installing cloudflared, and running it persistently. For production, we'll configure the tunnel as a systemd service and enforce strict TLS (HTTPS) for backend services.

Prerequisites

- Ubuntu 22.04 LTS installed and updated (sudo apt update && sudo apt upgrade -y).
- Docker installed and running with your app containers exposed on localhost ports (e.g., a web app on localhost:8080). Test locally with curl http://localhost:8080 or curl https://localhost:8443 (for HTTPS services).
- A Cloudflare account with a domain added and DNS managed by Cloudflare.
- Zero Trust enabled on your domain (go to dash.cloudflare.com > Zero Trust > Settings > Enable Zero Trust).
- Root or sudo access on the server.
- Your Docker services configured for HTTPS (self-signed or valid certs) for production security. Cloudflare will terminate TLS at the edge and re-encrypt to your origin.

Note: This setup uses dashboard-managed tunnels for simplicity via the web portal. All traffic will be HTTPS-only via Cloudflare's edge.

Step 1: Install cloudflared on Ubuntu 22.04

cloudflared is the daemon that creates the outbound tunnel from your server to Cloudflare.

1. Add the Cloudflare GPG key:

- ```
  sudo mkdir -p --mode=0755 /usr/share/keyrings
  ```

  ```
  curl -fsSL https://pkg.cloudflare.com/cloudflare-main.gpg | sudo tee /usr/share/keyrings/cloudflare-main.gpg >/dev/null
  ```

- Add the Cloudflare repository (use "jammy" for Ubuntu 22.04):

- ```
  echo 'deb [signed-by=/usr/share/keyrings/cloudflare-main.gpg] https://pkg.cloudflare.com/cloudflared jammy main' | sudo tee /etc/apt/sources.list.d/cloudflared.list
  ```

- Update the package list and install cloudflared:

- ```
  sudo apt update
  sudo apt install cloudflared
  ```
- Verify the installation:

4. ```
   cloudflared version
   ```
   You should see output like cloudflared version 2025.x.x.

Step 2: Create a Tunnel in the Cloudflare Zero Trust Dashboard

Use the web portal to create and manage the tunnel.

1. Log in to the Zero Trust dashboard and select your account/domain.
2. Navigate to **Networks > Tunnels**.
3. Click **Create a tunnel**.
4. Select **Cloudflared** as the connector type and click **Next**.
5. Enter a descriptive name for your tunnel (e.g., homelab-docker-prod) and click **Save tunnel**.
6. On the next screen, confirm the environment is **Linux/Ubuntu**, copy the provided installation/run command (it looks like cloudflared tunnel --url localhost:8000 run --token eyJh...), and paste it into your server's terminal. Run it now to test the connection.
   - This command authenticates and starts the connector. You'll see logs like "Registered tunnel connection" if successful.
   - Once connected, the dashboard will show the connector as **Healthy** under your tunnel. Click **Next**.

Note: Keep this terminal open for now; we'll make it persistent in Step 5.

Step 3: Configure Public Hostnames for Docker App Containers

Route traffic from your domain to your Docker containers. For HTTPS-only production, configure services as HTTPS and enable strict TLS verification.

1. In the tunnel details page, go to the **Public Hostname** tab (under **Published application routes**).
2. Click **Add a public hostname**.
3. Enter the **Subdomain** (e.g., myapp) and select your **Domain** from the dropdown (e.g., example.com). The full URL will be https://myapp.example.com.
4. Under **Service**:
   - **Type:** Select HTTPS (for HTTPS-only services; use HTTP only if your Docker app doesn't support TLS, but avoid for production).

    - **URL:** Enter the local address of your Docker container, e.g., localhost:8443 (replace with your app's port). Use https://localhost:8443 if specifying protocol.
5. Under **Additional application settings** (for production HTTPS hardening):
    - Set **TLS verify**: Full (strict) to enforce valid certificates on your origin server (recommended for production; requires valid certs on Docker apps).
    - Optionally, add **No TLS Verify** only for testing self-signed certs (not production).
    - Enable **HTTP/2** and **HTTP/3` for performance if supported by your apps.
6. Click **Save hostname**. Repeat for each Docker app (e.g., another for db.example.com pointing to localhost:5432 if applicable, but use appropriate ports).

Your services are now accessible at https://<subdomain>.<domain> with automatic HTTPS enforcement by Cloudflare. Test with curl https://myapp.example.com.

Step 4: Secure the Setup for Production (HTTPS-Only)

- **Edge TLS:** Cloudflare handles TLS termination; all public access is HTTPS-only by default.
- **Origin TLS:** With Full (strict), Cloudflare verifies your Docker app's cert. Generate certs via Let's Encrypt or self-signed (but use CA-trusted for strict mode).
- **Access Policies:** For added security, go to **Access > Applications** > Add an application > Select your hostname > Configure policies (e.g., require email login).
- **WAF and Rate Limiting:** Enable in **Security > WAF** for your domain to protect against attacks.
- **Docker-Specific:** Ensure containers bind to 0.0.0.0 or localhost only (not public interfaces). Use Docker Compose for multi-container setups.

Step 5: Run cloudflared as a Systemd Service for Production Persistence

The initial run command is temporary. For headless production, create a systemd service to auto-start on boot.

1. Stop the current cloudflared process (Ctrl+C in the terminal).
2. Create a config directory (optional, but good practice):

- `sudo mkdir -p /etc/cloudflared`
- Copy your token from the dashboard (under Tunnels > Your Tunnel > Connectors > Edit > Copy the run command's --token value). Save it securely as /etc/cloudflared/token.txt:
- ```
  sudo tee /etc/cloudflared/token.txt > /dev/null << EOF
  eyJhIjoi... (your full token)
  EOF
  sudo chmod 600 /etc/cloudflared/token.txt
  ```
  ```

- Create the systemd service file:

- ```
  sudo tee /etc/systemd/system/cloudflared.service > /dev/null << EOF
  [Unit]
  Description=Cloudflared Tunnel
  After=network.target
  Wants=network.target

  [Service]
  Type=notify
  ExecStart=/usr/bin/cloudflared tunnel run --token $(cat
  /etc/cloudflared/token.txt)
  Restart=always
  RestartSec=5

[Install]
  WantedBy=multi-user.target
  EOF
  ```

- Reload systemd, enable, and start the service:
  text
- ```
  sudo systemctl daemon-reload
  sudo systemctl enable cloudflared
  sudo systemctl start cloudflared
  ```

- Verify the service:

6. ```
   sudo systemctl status cloudflared
   ```

   Look for "Active: active (running)" and healthy connector in the dashboard. Logs: sudo journalctl -u cloudflared -f.

If issues arise, check logs for token errors or port conflicts.

Step 6: Verify and Test the Setup

1. In the dashboard, confirm the connector is **Healthy**.
2. Test access: From an external browser or curl https://myapp.example.com – it should proxy to your Docker app.
3. Monitor: Use **Networks > Tunnels > Metrics** for traffic stats.
4. Update: To upgrade cloudflared, run sudo apt update && sudo apt upgrade cloudflared, then sudo systemctl restart cloudflared.
   ```

Troubleshooting

- **Connector Offline:** Ensure firewall allows outbound UDP/443 to Cloudflare (no inbound ports needed). Check ufw status.
- **TLS Errors:** Verify Docker app certs; switch to Flexible TLS temporarily for testing.
- **Docker Not Reachable:** Confirm container ports with docker ps and test locally.
- **Logs:** sudo journalctl -u cloudflared or dashboard tunnel logs.

This setup provides a secure, zero-trust tunnel for your homelab Docker apps. For advanced configs (e.g., multiple services), edit public hostnames in the dashboard. If your Docker apps need custom paths, add them in the subdomain field (e.g., /api).

Here is a **step-by-step, screenshot-free elaboration** of exactly how to **copy the token from the Cloudflare Zero Trust dashboard** and **securely save it as /etc/cloudflared/token.txt** on your **Ubuntu 22.04 headless homelab server**.

Step 1: Locate Your Tunnel in the Dashboard

1. Open your browser and go to: https://one.dash.cloudflare.com
2. Select your **account** and **domain** (the one you added to Cloudflare).
3. In the left sidebar, click:

```
3. Networks → Tunnels
```

1. You will see your tunnel listed (e.g., homelab-docker-prod). Click on the **tunnel name** to open its details.

Step 2: Find the Connector and Copy the Token

1. In the tunnel details page, look for the **"Connectors"** section (usually near the top).
2. You should see one connector listed (e.g., connector-1 or a UUID).
3. Click the **three dots (⋯)** or **"Edit"** button next to the connector.
4. A modal or panel will open showing:
    - Connector name
    - Last connected time
    - **Run command** (this is key)

    Example:

- `cloudflared tunnel run --token eyJhIjoiMTIzNDU2Nzg5MGFiY2RlZiIsImEiOiI...`
- **Copy the entire --token value**, including the eyJ... part.

Do NOT copy the full command — only the token string after --token.

Example token (truncated for safety):

5. `eyJhIjoiMTIzNDU2Nzg5MGFiY2RlZiIsImEiOiIxMjM0NTY3ODkwYWJjZGVmIiwidCI6IjEyMzQtNTY3OC05MGFiLWNkZWYiLCJzIjoiMTIzNDU2Nzg5MGFiY2RlZiJ9`

   This token is **long**, **base64-encoded**, and **unique**. Treat it like a password.

Step 3: Save the Token Securely on Your Ubuntu Server

Now, SSH into your **headless Ubuntu 22.04 server** and run these commands **exactly**:

```
# Create the config directory (if not exists)
sudo mkdir -p /etc/cloudflared

# Write the token to a file using a secure method (heredoc)
sudo tee /etc/cloudflared/token.txt > /dev/null << 'EOF'
eyJhIjoiMTIzNDU2Nzg5MGFiY2RlZiIsImEiOiIxMjM0NTY3ODkwYWJjZGVmIiwidCI6IjEyMzQtNTY3OC05MGFiLWNkZWYiLCJzIjoiMTIzNDU2Nzg5MGFiY2RlZiJ9
EOF
```

Replace the token above with the **actual full token** you copied from the dashboard.

Step 4: Secure the File (Critical for Production)

```
# Set strict permissions: only root can read
sudo chmod 600 /etc/cloudflared/token.txt

# Verify ownership
sudo chown root:root /etc/cloudflared/token.txt
```

Check it:

```
ls -la /etc/cloudflared/token.txt
```

Expected output:

```
-rw------- 1 root root 256 Oct 25 12:34 /etc/cloudflared/token.txt
```

Step 5: Verify the Token Works (Optional but Recommended)

Test it manually before using in systemd:

```
cloudflared tunnel run --token $(cat /etc/cloudflared/token.txt)
```

You should see:

```
INF Registered tunnel connection connIndex=0 ...
```

Press Ctrl+C to stop — this confirms the token is valid.

Security Notes

Risk	Mitigation
Token exposed in shell history	Use tee with heredoc (as shown) — avoids history
File readable by non-root	chmod 600 + chown root:root
Token in logs	Never run cat or echo the token in scripts

Final File Location & Permissions

```
/etc/cloudflared/
└── token.txt    ← contains only the raw token, nothing else
```

Never commit this file to Git, backup it securely, and rotate if compromised.

You are now ready to proceed to **Step 5** of the main tutorial: creating the cloudflared.service systemd unit that reads this token securely.

To **add another Docker app (new subdomain)** to your **existing Cloudflare Tunnel**, you **only need to replicate one section** from the original tutorial:

Replicate Step 3: Configure Public Hostnames for Docker App Containers

> **Do NOT repeat installation, token creation, or systemd setup.**
>
> Your cloudflared tunnel is already running and connected — just **add more routes**.

Step-by-Step: Add a New Docker App (e.g., wiki.example.com)

1. **Log in to Cloudflare Zero Trust Dashboard**

   → https://one.dash.cloudflare.com

2. Go to:

- `Networks → Tunnels → [Your Tunnel Name: homelab-docker-prod]`
- Click the **Public Hostname** tab.
- Click **Add a public hostname**
- Fill in the form:

Field	Value
Subdomain	wiki (or grafana, nextcloud, etc.)
Domain	example.com (your domain)
Service Type	HTTPS (or HTTP if not TLS-enabled)
URL	localhost:3000 ← **your new Docker app's port**
TLS verify	Full (strict) (for production)
No TLS Verify	Only for testing self-signed certs

  - Example: Grafana on port 3000 → https://grafana.example.com → localhost:3000

- Click **Save hostname**
- **Wait 10–30 seconds** → Tunnel auto-updates. No restart needed.
- **Test externally**:

8. `curl -I https://grafana.example.com`

   Should return 200 OK or redirect to login.

Summary: What You DO NOT Repeat

Step	Do NOT Repeat
Install cloudflared	Already done
Create tunnel in dashboard	Already exists
Copy token / create /etc/cloudflared/token.txt	Already done
Create cloudflared.service	Already running

Pro Tip: Organize Multiple Apps

Subdomain	Docker Port	App
app.example.com	localhost:8080	Main web app
grafana.example.com	localhost:3000	Monitoring
wiki.example.com	localhost:8081	Wiki
files.example.com	localhost:5000	File browser

All share **one tunnel**, **one cloudflared process**, **zero open ports**.

Done. Your new Docker app is now securely accessible via HTTPS with no config changes on the server.

TL;DR: Safe to Use — But Configure Correctly

Tool	Safe?	Required Change
ufw	YES	**Allow outbound UDP/443** (Cloudflare Tunnel)
fail2ban	YES	**Disable inbound SSH/Docker jails** or **whitelist Cloudflare IPs**
crowdsec, ossec, etc.	YES	Same logic as fail2ban

> **Cloudflare Tunnel = 100% outbound-only** → No inbound ports → Nothing to block on your firewall.

How Cloudflare Tunnel Works (Security Implications)

```
Your Server (Homelab)
       |
```

```
└──► cloudflared → Outbound UDP/443 → Cloudflare Edge
                                    (4 locations: AWS, etc.)
                                    |
                              Public Internet (HTTPS only)
```

- **No inbound ports open** (22, 80, 443, Docker ports, etc.)
- **All traffic originates from your server**
- **Cloudflare IPs → your server (via tunnel)** → never direct public access

---

1. ufw (Uncomplicated Firewall) — Safe & Recommended

What to Allow (Minimal Rules)

```
sudo ufw allow out 443/udp   # Cloudflare Tunnel (required)
sudo ufw allow out 53       # DNS (optional but good)
sudo ufw allow 22/tcp       # SSH (only if you need remote access)
```

Recommended: Deny All Inbound (Default)

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw enable
```

Result:

- Tunnel works (outbound UDP/443)
- SSH works (if allowed)
- **No public services exposed** → Docker ports safe even if bound to 0.0.0.0

---

2. fail2ban — Safe, But Avoid False Bans

Problem: fail2ban Monitors Logs for Failed Logins

Service	Risk
SSH (sshd)	**High** — Cloudflare health checks or WARP clients may trigger bans
Docker apps (e.g., Nextcloud, Traefik)	**Medium** — brute-force attempts via tunnel → banned → but tunnel still up

Solution: Whitelist Cloudflare IPs in fail2ban

Step 1: Create Cloudflare IP List

```
sudo mkdir -p /etc/fail2ban
sudo curl -L https://www.cloudflare.com/ips-v4 -o /etc/fail2ban/cloudflare-ips.txt
sudo curl -L https://www.cloudflare.com/ips-v6 -o - >> /etc/fail2ban/cloudflare-ips.txt
```

Step 2: Add to jail.local

```
sudo nano /etc/fail2ban/jail.local
```

```
[DEFAULT]
ignoreip = 127.0.0.1/8 ::1
# Whitelist Cloudflare
bantime = 1h
findtime = 10m
maxretry = 5

# Load Cloudflare IPs
ignorecommand = grep -qF -f /etc/fail2ban/cloudflare-ips.txt %(ip)s && exit 1 || exit 0
```

Step 3: Restart fail2ban

```
sudo systemctl restart fail2ban
```

Result:

- Real attackers (via tunnel) → banned at origin (your Docker app)
- Cloudflare edge traffic → never banned
- SSH safe from Cloudflare health checks

---

3. Other Security Tools — Safe with Caveats

Tool	Recommendation
CrowdSec	Use cloudflare bouncer → auto-whitelist
OSSEC / Wazuh	Monitor logs normally — no IP blocking needed
Docker Security	Bind apps to localhost only → no external exposure

Tool	Recommendation
Rootkit Hunters	Run freely — no interference

Best Practices Summary

Action	Why
ufw: Allow **outbound UDP/443**	Tunnel needs this
ufw: Deny all inbound	No public services
fail2ban: Whitelist Cloudflare IPs	Prevent false bans
Docker: Bind to localhost	Even if app listens on 0.0.0.0, tunnel only allows via localhost:<port>
Use **HTTPS + Full (strict)** in tunnel	Enforce TLS end-to-end

---

Verify Your Setup

```
# 1. Check UFW
sudo ufw status verbose

# 2. Test Tunnel (should work)
curl -I https://yourapp.example.com

# 3. Test fail2ban ignore
sudo fail2ban-client get sshd ignoreip | grep -f /etc/fail2ban/cloudflare-ips.txt
```

Final Verdict

> **Run ufw + fail2ban — they improve security.**
>
> **Just configure them to respect the outbound-only tunnel model.**

You're not exposing anything — you're **hardening the origin** while Cloudflare handles the edge.

No risk of blocking legitimate access if you follow the whitelist rules above.